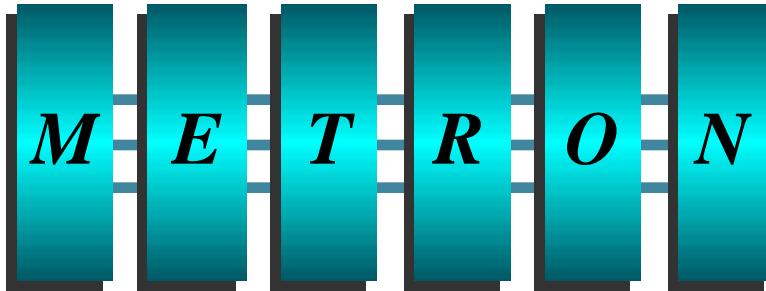


# *Object Proxies*

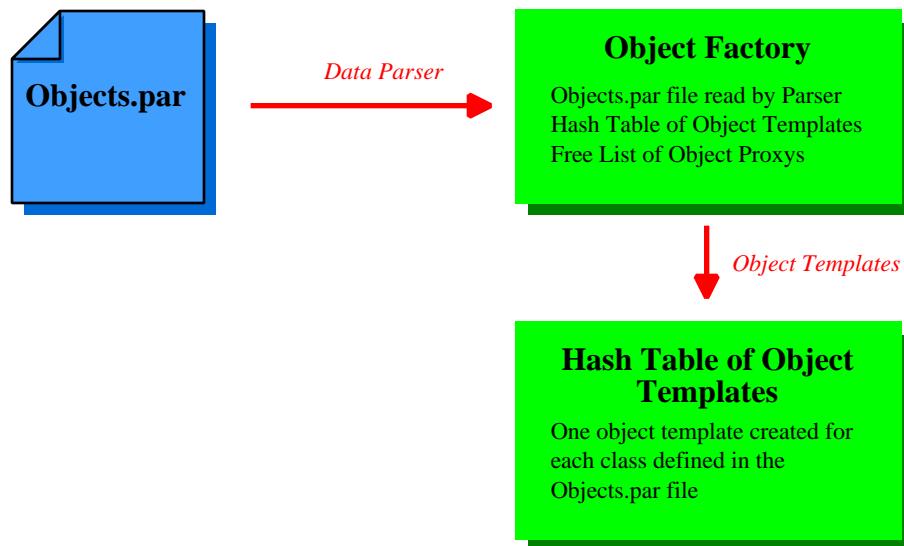
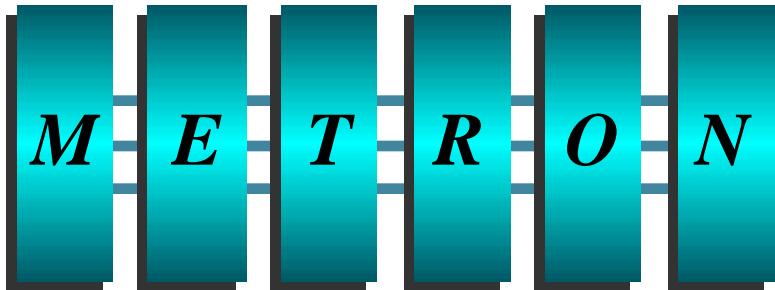
**Dr. Jeffrey S. Steinman  
Metron Incorporated  
December 16, 1996**

# Motivation for Object Proxies



- **Optimistic parallel simulations permit events to operate only on a single object because:**
  - Objects are distributed
  - Objects can be at different times
  - Rules must be followed for supporting rollback
- **HLA needs to transport an entity's published attributes**
  - Attributes defined in an input file (RID)
- **Automation of Data Distribution Management**
  - Delivering attributes as they are updated
  - Update and subscription regions
  - Dynamic attributes

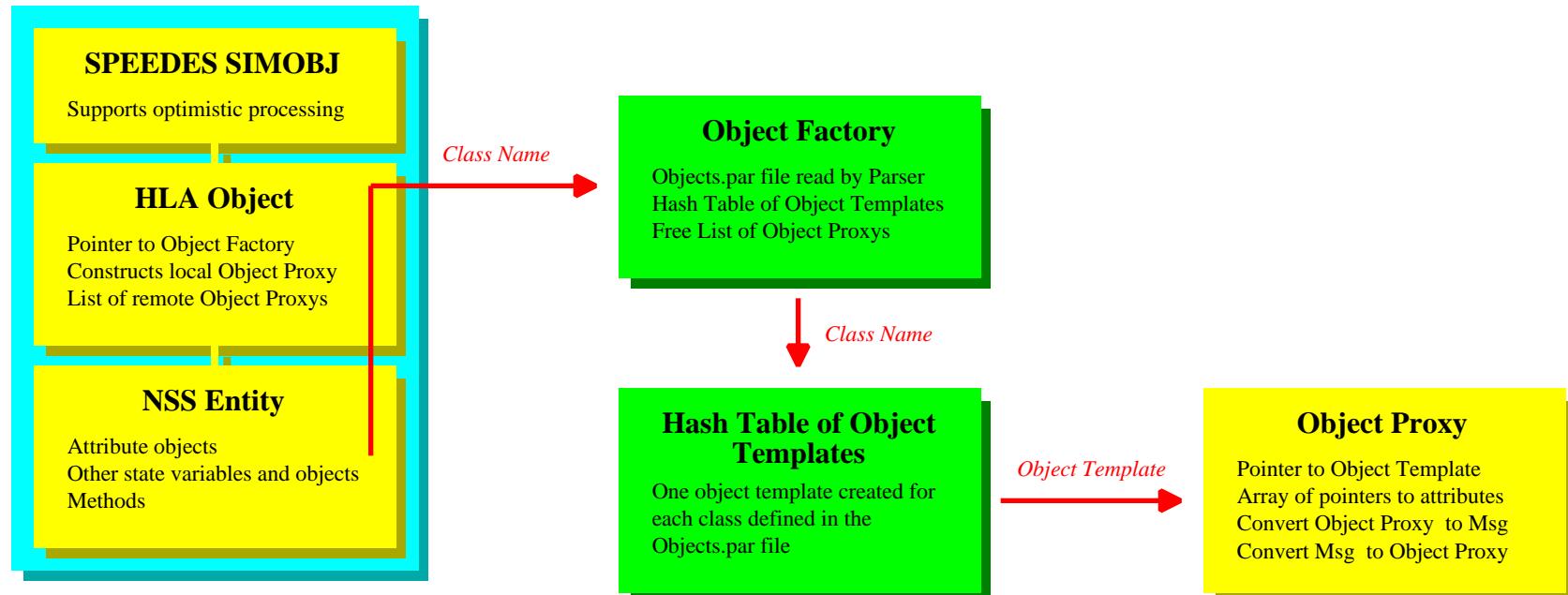
# *Creating the Object Factory*



1. The Objects.par file contains all of the class descriptions for Object Proxies
2. A single Object Factory on each node uses the parser to read Objects.par
3. The Object Factory creates an Object Template for each class description
4. The Object Templates are stored in a hash table for future lookups

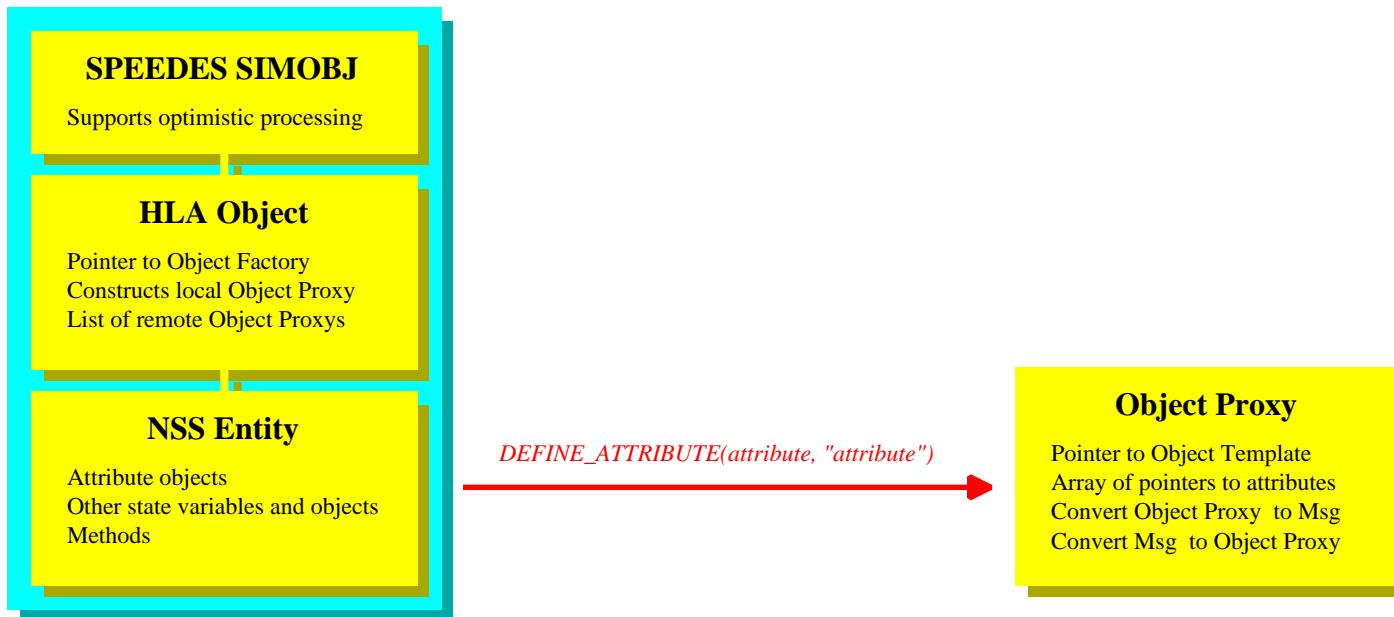
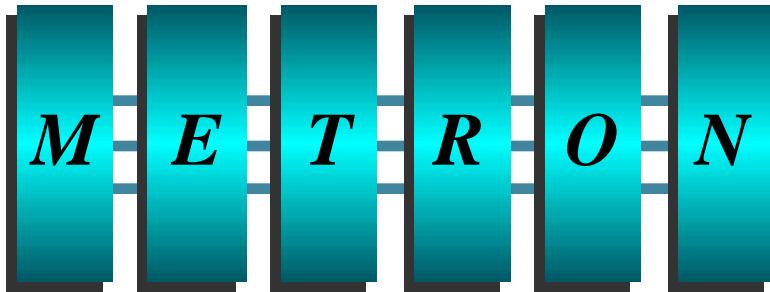
# *Creating an Object Proxy*

M E T R O N



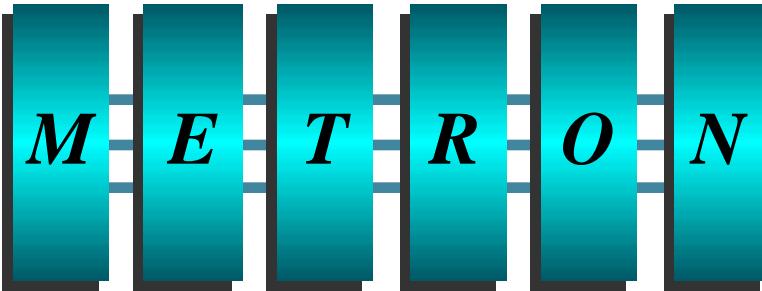
1. Each NSS Entity passes its class name to its base-class HLA Object during the constructor
2. The base-class HLA Object then passes the class name to the Object Factory
3. The Object Factory obtains the corresponding Object Template and uses it to create the Object Proxy
4. The Object Proxy is created and is given a pointer back to its corresponding Object Template
5. The NSS Entity now has a pointer to its Object Proxy

# *Mapping Attributes to an Object Proxy*



1. Each NSS Entity contains ATTRIBUTE objects possibly mixed with other state variables
2. The ATTRIBUTES may reside in contained or inherited objects
3. The **DEFINE\_ATTRIBUTE** Macro maps addresses of attributes to the Object Proxy's attribute array
4. The ATTRIBUTE objects overload operators to automate distribution of attributes when they change

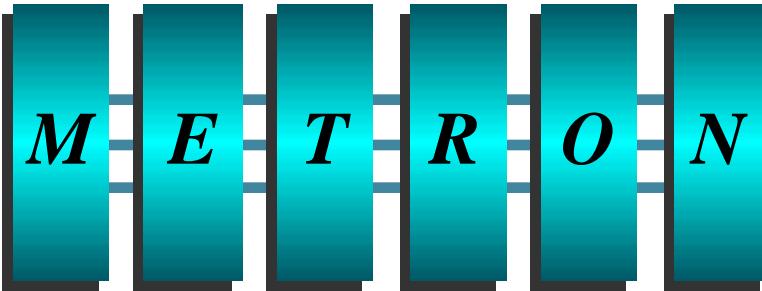
# Object Proxy Key Elements



- **Object Proxy**

- Represents the "published" attributes of an instantiated entity
- Can be packaged into a message that is distributed to other processors
- Can be a base-class used with inheritance
  - *Provides normal and virtual methods*
  - *Can provide strong compiler checking on "get" methods*
  - *Can store other "non-distributed" data*
- Uses special C++ attribute objects to automate distribution when attributes are updated
  - *Static: INT, FLOAT, LOGICAL, STRING, POSITION*
  - *Dynamic: INT, FLOAT, LOGICAL, POSITION*
  - *Contained objects*
  - *Lists of objects*

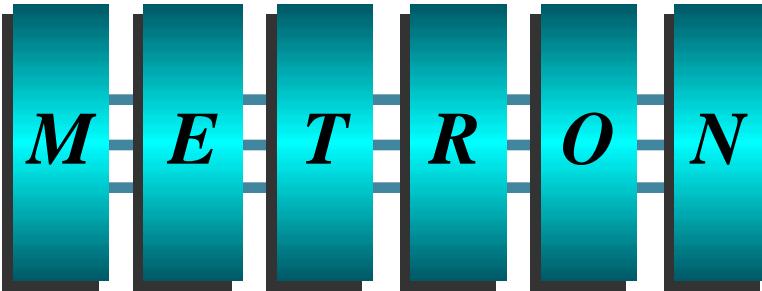
# Object Proxy Key Elements



- **HLA Object**

- NSS Entities inherit from it
  - *Interface is direct (method calls, operator overloading, Macros, etc.)*
- HLA Entities map to it
  - *Interface is indirect (HLA interfaces)*
- Each HLA Object has an Object Proxy to represent its attributes and a list of Object Proxies for other objects that pass its filter
- DDM is automated through events that operate at the HLA Object level and not at the NSS entity level

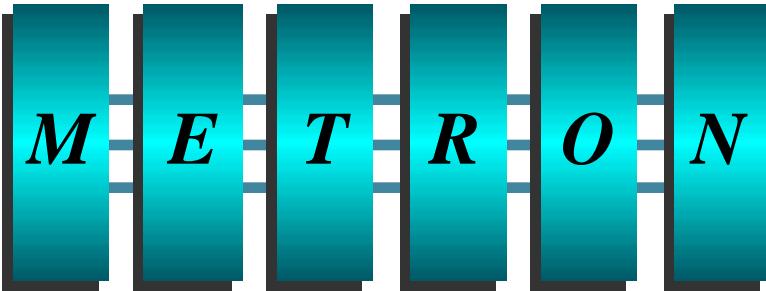
# Object Proxy Key Elements



## • Object Factory

- One Object Factory per node
  - *Available to HLA Object and to Contained Object Attributes*
- Reads the Objects.par file using the Parser
  - *Describes attributes of the different classes of objects*
  - *Creates Object Templates for each object class and stores them in a hash table*
- Manages a free list for each class of Object Proxy for fast memory management

# Object Proxy Key Elements

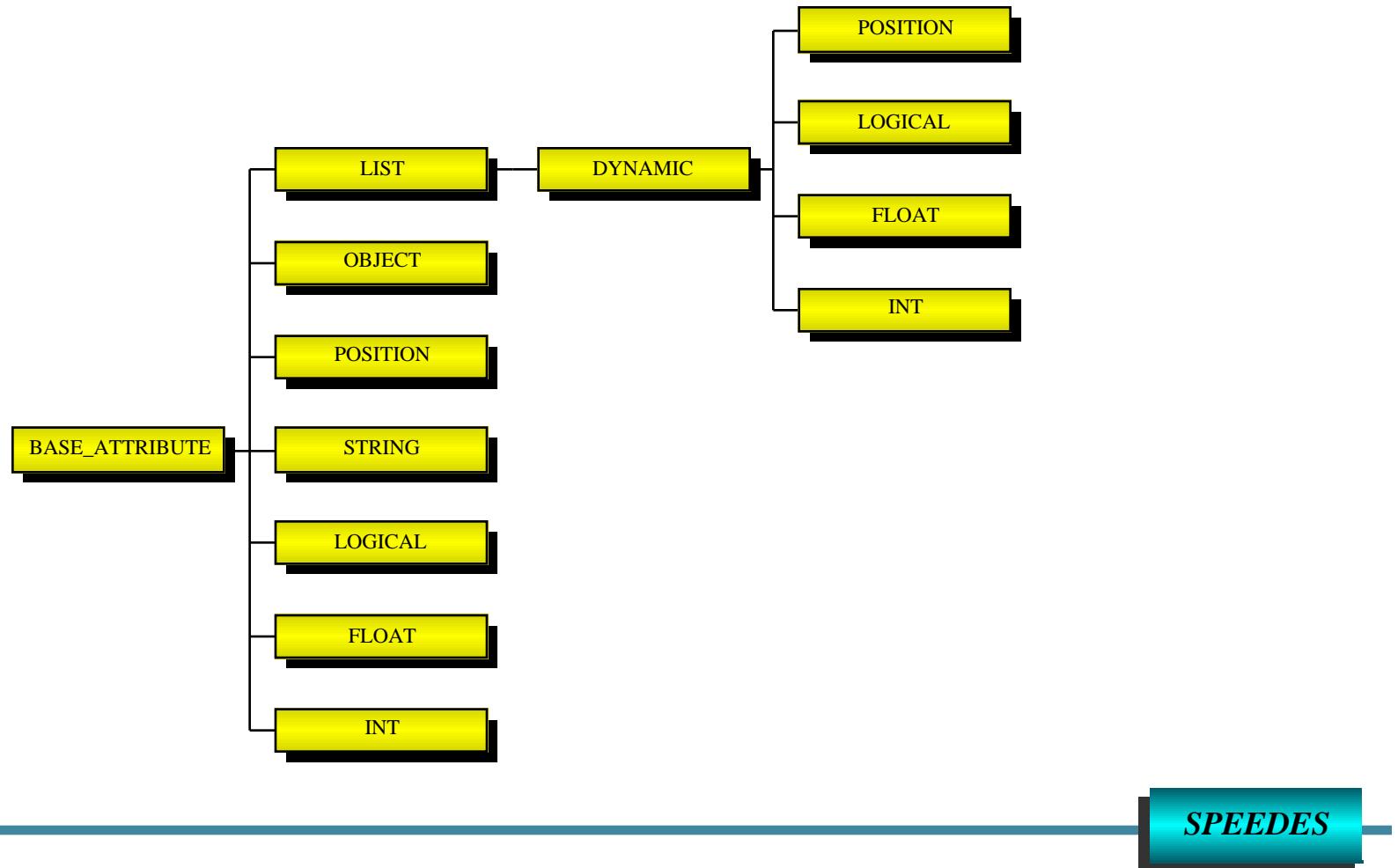


- **Object Template**

- Stores object class information obtained from the Objects.par file
  - *Defines integer Ids for each attribute in each object class*
- One Object Template is created per node for each class
  - *Object Templates are stored in a hash table and are references by instances of Object Proxies*
- Supports inheritance (but not multiple)
  - *Supports arbitrary levels of contained objects*
- Stores virtual methods for classes
  - *Virtual methods can have arbitrary calling parameters*
  - *Pointers to methods are not passed with Object Proxies since they are referenced in the Object Templates*
- Supports arrays of attributes (of any type)

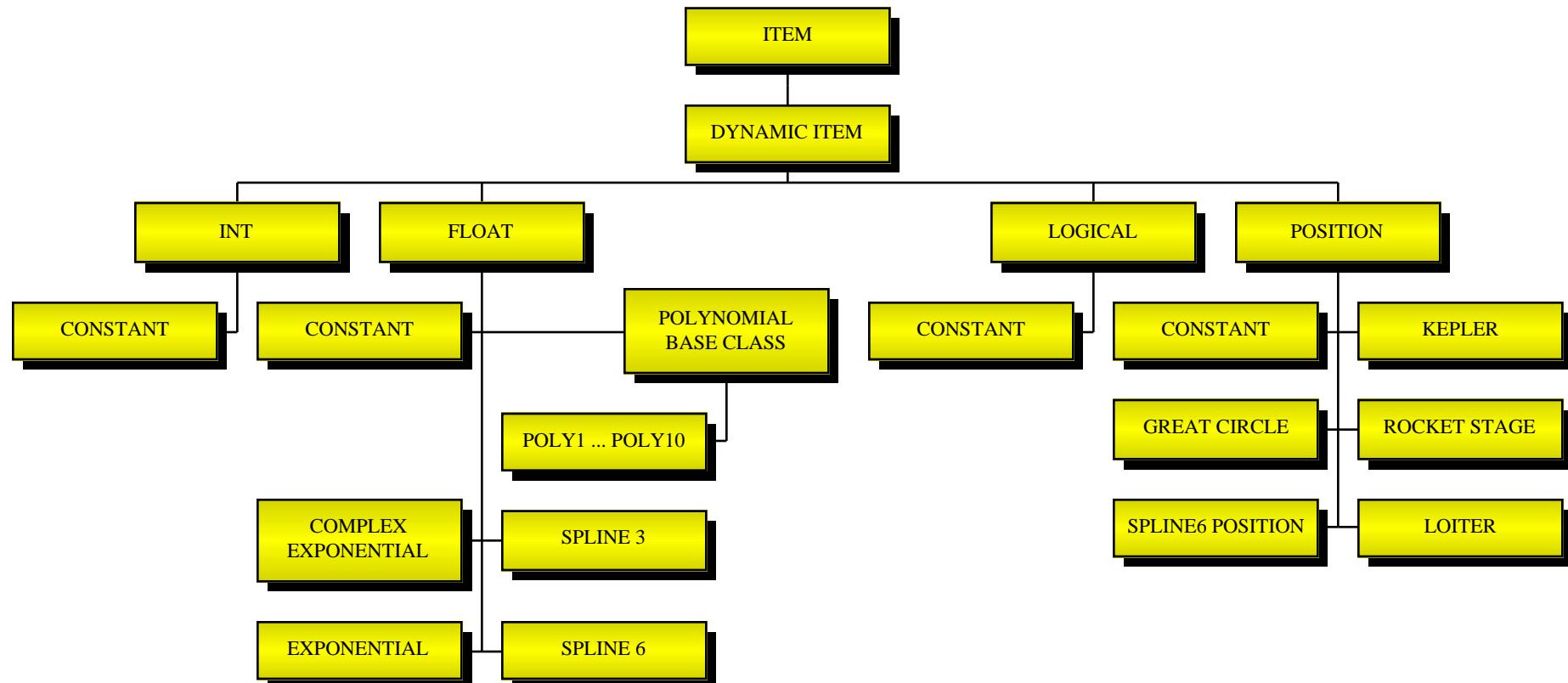
M E T R O N

## Attribute Object Inheritance Tree



# *Dynamic Items Inheritance Tree*

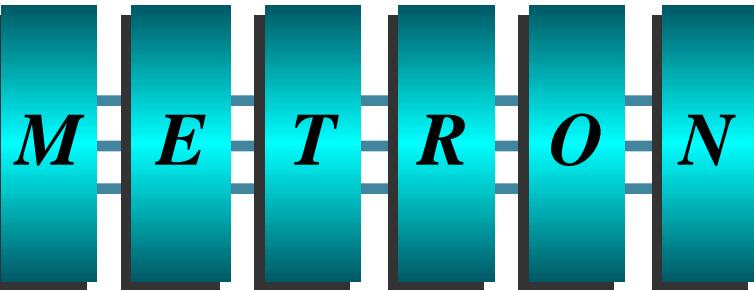
M E T R O N



**SPEEDES**

# *Objects.par to C++*

## *Class Mapping*



```

ENTITY {
    define logical alive
    define dynamic_position Position
}

AIRCRAFT {
    reference INHERIT ENTITY
    define string TailNumber
    define int Nengines
    define dynamic_float fuel
    VIRTUAL METHODS {
        define void Print
        define void GetSpeed
    }
}

F18 {
    reference INHERIT AIRCRAFT
    define object RADAR
    VIRTUAL METHODS
        define void Print
    }
}

RADAR {
    define float RadarRange
    define float BeamWidth
    define float Frequency[3]
}

```

Objects.par

```

class ENTITY : public HLAOBJ {
    LOGICAL_ATTRIBUTE alive;
    DYNAMIC_POSITION Position;
    ENTITY(char *ClassType);
};

class AIRCRAFT : public ENTITY {
    STRING_ATTRIBUTE TailNumber;
    INT_ATTRIBUTE Nengines;
    DYNAMIC_FLOAT_ATTRIBUTE fuel;
    AIRCRAFT(char *ClassType);
};
void AIRCRAFT_PRINT(void *ObjectProxy, va_list &va);
void AIRCRAFT_GetSpeed(void *ObjectProxy, va_list &va);

class F18 : public AIRCRAFT {
    OBJECT_ATTRIBUTE RADAR;
    F18(char *ClassType);
};
void F18_PRINT(void *ObjectProxy, va_list &va);

RADAR {
    FLOAT_ATTRIBUTE RadarRange;
    FLOAT_ATTRIBUTE BeamWidth;
    FLOAT_ATTRIBUTE Frequency[3];
    RADAR();
};

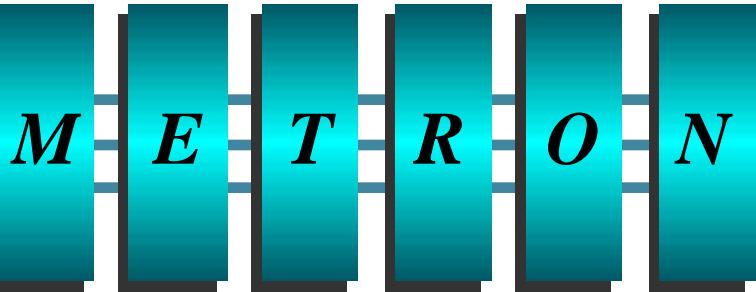
```

C++

SPEEDES

# Sample Application

## C++ Code



```

ENTITY::ENTITY(char *ClassType) : HLAOBJ (ClassType) {
    DEFINE_ATTRIBUTE(alive, "alive");
    DEFINE_ATTRIBUTE(Position, "Position");
    alive = LOGICAL_TRUE;
    Position.SetEARTH(32.0, 43.0, 0.0);
}

AIRCRAFT::AIRCRAFT(char *ClassType) : ENTITY(ClassType) {
    DEFINE_ATTRIBUTE(TailNumber, "TailNumber");
    DEFINE_ATTRIBUTE(Nengines, "Nengines");
    DEFINE_ATTRIBUTE(fuel, "fuel");
    DEFINE_METHOD(AIRCRAFT_PRINT, "Print", "AIRCRAFT");
    DEFINE_METHOD(AIRCRAFT_GetSpeed, "GetSpeed");
    TailNumber = "Hornet001";
    Nengines = 2;
    DYNAMIC_EXPONENTIAL *DE = (DYNAMIC_EXPONENTIAL *)
        FreeList->new_object(DYNAMIC_EXPONENTIAL_ID);
    DE->SetStartTime(0.0); DE->SetEndTime(3600.0);
    DE->SetTimeConstant(1800.0); DE->SetAmplitude(1000.0);
    fuel += DE;
    DYNAMIC_SPLINE_3 *DS3 = (DYNAMIC_SPLINE_3 *)
        FreeList->new_object(DYNAMIC_SPLINE_3_ID);
    DS3->init(0.0, 1000.0, 1.0, 3600.0, 500.0, 0.5);
    fuel += DS3;
}

F18::F18(char *ClassType) : AIRCRAFT(ClassType) {
    DEFINE_ATTRIBUTE(RADAR, "RADAR");
    DEFINE_METHOD(F18_PRINT, "Print", "F18");
}

```

```

RADAR::RADAR() {
    SetClassName('RADAR');
    DEFINE_ATTRIBUTE(RadarRange, "RadarRange");
    DEFINE_ATTRIBUTE(BeamWidth, "BeamWidth");
    DEFINE_ATTRIBUTE(Frequency, "Frequency");
    RadarRange = 80.0;
    BeamWidth = 0.005;
    Frequency[0] = 100.0; Frequency[1] = 200.0; Frequency[2] = 300.0;
}

void AIRCRAFT_PRINT(void *ObjectProxyHandle, va_list &va) {
    OBJECT_PROXY* OP = (OBJECT_PROXY *)ObjectProxyHandle;
    ostream *out = va_arg(va, ostream *);
    *out << "AIRCRAFT_PRINT" << endl; OP->PrintStream(*out);
}

void AIRCRAFT_GetSpeed(void *ObjectProxyHandle, va_list &va) {
    OBJECT_PROXY* OP = (OBJECT_PROXY *)ObjectProxyHandle;
    double time = va_arg(va, double);
    double *speed = va_arg(va, double *);
    double X[3], V[3];
    OP->GetPosVel(X, V, time);
    speed = sqrt(V[0]*V[0]+V[1]*V[1]+V[2]*V[2]);
}

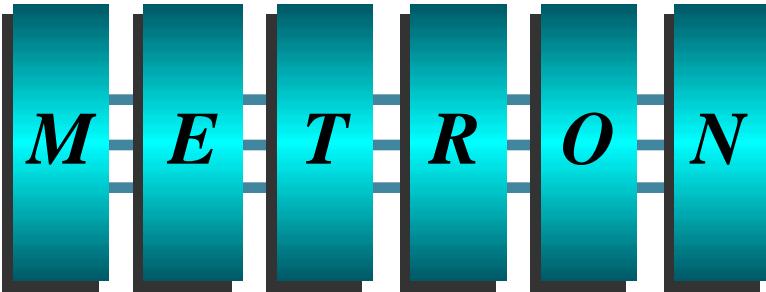
void F18_PRINT(void *ObjectProxyHandle, va_list &va) {
    OBJECT_PROXY* OP = (OBJECT_PROXY *)ObjectProxyHandle;
    ostream *out = va_arg(va, ostream *);
    *out << "F18_PRINT" << endl; OP->PrintStream(*out);
}

```

SPEEDES

# Sample Application

## C++ Code Continued



```

main() {
    F18 *TestObj = new F18("F18");
    double speed;
    double x[3];

    OBJECT_PROXY *ObjectProxy = TestObj->GetObjectProxy();

    ObjectProxy->PrintStream(cout);
    ObjectProxy->Execute("Print", cout);
    ObjectProxy->Execute("GetSpeed", 100.0, &speed);

    cout << "Nengines " << ObjectProxy->GetInt("Nengines") << endl;
    cout << "fuel at time 100 "
        << ObjectProxy->GetDynamicFloat("fuel", 100) << endl;

    ObjectProxy->GetPosition("Position", ECI, 3600.0, x);
    cout << "ECI position at time 36000 ["
        << x[0] << ", " << x[1] << ", " << x[2] << "]" << endl;

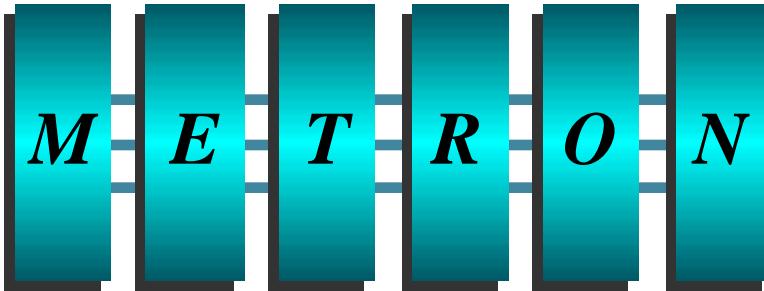
    int RefAlive = ObjectProxy->GetReference("Alive");
    cout << "Alive flag " << ObjectProxy->GetLogical(RefAlive) << endl;

    OBJECT_PROXY *ObjectProxy1 = ObjectProxy->GetObject("RADAR");
    cout << "RADAR Frequency[1] "
        << ObjectProxy1->GetFloat("Frequency", 1) << endl;
}

```

SPEEDES

# *Status of Software Components*



- Software Components
  - ▶ Objects.par input file format
    - Correctly read in by SPEEDES general-purpose Data Parser
  - ▶ HLA Object
    - NSS entities inherit from this base class - HLA entities map
    - Creates the entity's Object Proxy based on its class type
    - Supports all DDM-related functions
  - ▶ Object Factory
    - Hash table of object templates
    - Free list and user inheritance by end of January
  - ▶ Object Template
    - Attribute-to-index mapping
    - Inheritance and arbitrary levels of contained objects
    - Virtual Methods
  - ▶ Object Proxy
    - DEFINE\_ATTRIBUTE macro for mapping attributes
    - Object Proxy two-way message conversion by end of January
  - ▶ Attributes
    - Operator overloading to automate attribute distribution and rollbacks
    - Static INT, FLOAT, LOGICAL, STRING, POSITION
    - Dynamic: INT, FLOAT, LOGICAL, POSITION
    - OBJECTS and LISTS of OBJECTS

*Will be  
Fully  
Completed  
by end of  
January,  
1997*